PB84-149251

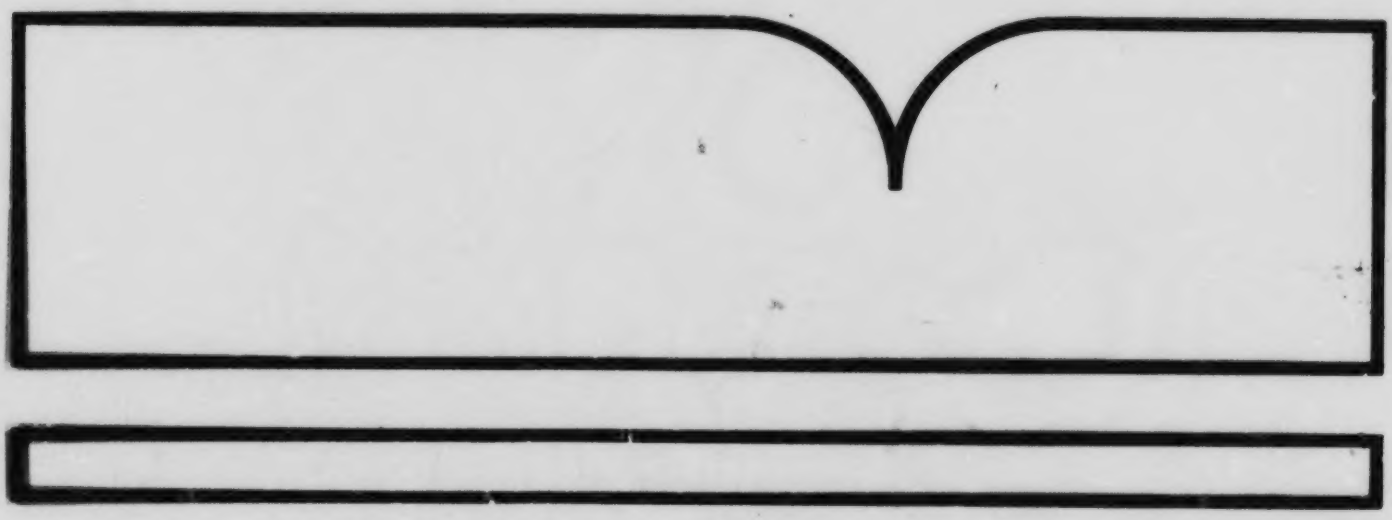
DATMAN FORTRAN User's Guide
Version 3.0

Rhode Island Univ., Kingston

Prepared for

Environmental Research Lab., Narragansett, RI

1984



U.S. Department of Commerce
National Technical Information Service

PB84-149251

EPA-600/D-84-033
January 1984

DATMAN FORTRAN USER'S GUIDE
VERSION 3.0

by

L. J. Bass
University of Rhode Island
Dept. of Computer Science and Experimental Statistics
Kingston, RI 02881

EPA CR 807031

EPA Project Officer
R. R. Payne

ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
NARRAGANSETT, RI 02882

REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

TECHNICAL REPORT DATA
(Please read Instructions on the reverse before completing)

1. REPORT NO.	2.	3. RECIPIENT'S ACCESSION NO.
EPA-600/D-84-033		PB84 149251

4. TITLE AND SUBTITLE	5. REPORT DATE
DATMAN FORTRAN User's Guide Version 3.0	1984
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT NO.
L.J. Bass	ERLN-X95
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Rhode Island Dept. of Computer Science and Experimental Statistics Kingston, Rhode Island 02881	10. PROGRAM ELEMENT NO. C2NN1E 11. CONTRACT/GRANT NO. CR 807031
12. SPONSORING AGENCY NAME AND ADDRESS Environmental Protection Agency Environmental Research Laboratory South Ferry Road Narragansett, Rhode Island 02882	13. TYPE OF REPORT AND PERIOD COVERED Technical Report 14. SPONSORING AGENCY CODE EPA/600/05

15. SUPPLEMENTARY NOTES

16. ABSTRACT

DATMAN is a data management system which runs on a variety of minicomputers. Currently, versions are supported on the following computers: PRIME and PDP 11/70 under IAS. DATMAN has facilities for creating data bases, adding, deleting, and editing data in data bases, retrieving selected data from data bases and processing the retrieved data. Access through DATMAN is either through a terminal interface with which the user executes DATMAN directly or through a FORTRAN interface within which the user calls FORTRAN subroutines to access DATMAN data bases. This manual is intended to describe the FORTRAN user interface.

17. KEY WORDS AND DOCUMENT ANALYSIS

a. DESCRIPTORS	b. IDENTIFIERS/OPEN ENDED TERMS	c. COSATI Field/Group

18. DISTRIBUTION STATEMENT	19. SECURITY CLASS *(This Report)*	21. NO. OF PAGES
Release to public	Unclassified	102
	20. SECURITY CLASS *(This page)*	22. PRICE
	Unclassified	

EPA Form 2220-1 (9-73)

NOTICE

This document has been reviewed in accordance with U.S. Environmental Protection Agency policy and approved for publication. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

TABLE OF CONTENTS

ABSTRACT.....1

INTRODUCTION.....2

I. OVERVIEW

DATMAN Data Bases.....I-1

Searchable Files.....I-3

Defining Rows by Ranges.....I-5

JOINing Data Bases.....I-8

Data Base of Names.....I-10

Data Independence.....I-11

Use of DATMAN Subroutines.....I-12

Modification of a Data Base.....I-15

Retrieving Data from a Data Base.....I-20

II. REFERENCE

Dimensioning DATMAN Arrays.....II-1

DBINIT.....II-4

DBOPN.....II-4

DBNAME.....II-5

DBCLOS.....II-7

DBADD.....II-8

DBCHNG.....II-9

DBDELT.....II-12

DBFIND.....II-14

DBGET.....II-19

DBJOIN.....II-20

DBORDR.....II-23

DBREWD.....II-24

DBSCAL.....II-24

DBINDX.....II-26
DBLENG.....II-27
DBTYPE.....II-28
DBVEX.....II-29
DBVINS.....II-30
DBVINT.....II-31
DBBLD.....II-32
FMTBLD.....II-36
SFBLD.....II-39

III. SAMPLE PROGRAMS

Data Base ANALYT.....III-2
Example Program 1.....III-3
Example Program 2.....III-5
Example Program 3.....III-7
Example Program 4.....III-9
Example Program 5.....III-11

IV. APPENDIX

A. Data Structure of Data Base.....IV-1
B. Accessing Subroutines on PRIME.....IV-4
C. Accessing Subroutines on PDP 11/70.....IV-5
D. List of Subroutines and Functions.....IV-6

ABSTRACT

DATMAN is a data management system which runs on a variety of minicomputers. Currently, versions are supported on the following computers: Prime and PDP 11/70 under IAS. DATMAN has facilities for creating data bases, adding, deleting, and editing data in data bases, retrieving selected data from data bases and processing the retrieved data. Access through DATMAN is either through a terminal interface with which the user executes DATMAN directly or through a FORTRAN interface within which the user calls FORTRAN subroutines to access DATMAN data bases. This manual is intended to describe the FORTRAN user interface.

The FORTRAN interface to DATMAN is intended for those applications for which the terminal interface is not adequate and assumes more sophistication on the part of the user than does the terminal interface. The FORTRAN interface also allows simultaneous access to multiple data bases. This permits much greater flexibility in the structuring of data bases than is allowed in the terminal interface.

The FORTRAN interface and the terminal interface are completely compatible. Any DATMAN data base can be accessed or modified by either the terminal interface or the FORTRAN interface. This allows the user of the DATMAN system to tailor an application using both the FORTRAN interface and the terminal interface as is most appropriate.

The initial sections of this quide provide a general overview of the factors involved in using the FORTRAN interface. This is followed by a list of the subroutine calls available with their calling sequences. The guide ends with several appendices giving sample programs using the FORTRAN interface.

Introduction

The FORTRAN user's guide is intended for those more sophisticated users of DATMAN who wish to write programs to access one or more DATMAN data bases. The FORTRAN interface is intended to complement the terminal interface and allow customizing of particular applications. It is assumed the reader has some familiarity with DATMAN and the terminal interface.

All of the data management functions of the terminal interface are available through FORTRAN subroutine calls. This includes calls to construct a data base, add or delete data from a data base, modify existing values in a data base, and retrieve data from a data base.

Additionally, the FORTRAN interface allows the writing of programs that are data independent. That is, programs which do not have to be modified when the format of the data being accessed is changed. Taking advantage of the data independence offered by DATMAN means that less labor will be needed to modify programs. Taking advantage of the data management and retrieval capabilities of DATMAN allows programmers to focus on the function of the program being written and worry less about the mechanics of getting and subsetting the data being manipulated.

This manual has an overview section describing DATMAN data bases, explaining data independence, and describing how the subroutine calls are used. The second section gives specific calling sequences for each subroutine and the last section gives sample programs demonstrating the use of the DATMAN FORTRAN interface.

SECTION I - OVERVIEW

DATMAN Data Base

A DATMAN data base can be viewed as a named table of data. The name of the table is used to identify the table for any future reference. The columns of the table have names as headings. These names (called variable names) are the only mechanism available to reference particular columns. When a data base is created each variable is assigned a type. Each data item for a particular variable is interpreted according to that type. The allowable types are characters (stored internally in ASCII), integer (stored internally as 4 byte integer) and real (stored internally as 8 byte real).

The rows of the table contain the data items. Within a particular row a variable can either have a value or be undefined. Undefined values are allowed and are supported by the system but they manifest themselves in various undesirable ways if the user is not aware of the possibility of undefined values.

No two rows of the table can be identical. In order to edit a row of data, the row to be changed must be identified. Therefore, some mechanism must be available to uniquely identify the row to be modified. In DATMAN, certain variables are specified by the user when the data base is created as being the variables to use to uniquely identify a row of data. These identification variables are called primary index variables. In some applications one variable is sufficient to

uniquely identify a row, in other applications it may take more than one. Variables which are designated as primary index variables may not have undefined values.

For retrieval purposes it is desirable to be able to retrieve rows of a data base by the value of a variable. For example, retrieve the row with the value of 10 for variable VAR1. In order to allow for this accessing by value, a variable can be designated as an index variable. For these variables, DATMAN will maintain the additional structures necessary to allow this accessing by variable.

Within a DATMAN data base, then, are two special types of variables. The first type, primary index variables, are used to uniquely identify a row and DATMAN maintains the integrity of these variables. DATMAN will not allow the existence of two rows with the same values of the primary index variables. The second type of special variable, index variables, are used to allow for accessing a data base by value. Primary index variables are automatically secondary index variables.

Within DATMAN, there are certain constraints on the length of variable names and the length of character strings. The restrictions are:

1) Variable names are no more than 10 characters long. They must start with a letter, A-Z, and allowable characters are A-Z, 0-9, $, and _.
2) No more than 7 primary index variables.
3) Character variables which are not index variables have a maximum length of 80 characters.
4) Character variables which are index variables have a maximum length of 15 characters.

Variables take up this space depending upon their type. Integers take up 4 characters, real numbers take up 8 characters and character variables take up the number of characters designated as their length.

Searchable Files

It is possible to take advantage of some of the retrieval features of DATMAN for fixed format ASCII files and also to use DATMAN's interactive input capabilities without constructing a DATMAN data base. These uses are accomplished by the use of the "searchable file" structure of DATMAN.

A searchable file is created by a call to SFBLD and it consists of two portions. One portion is the data file. This is either the name of a fixed format ASCII file or the characters TERM. The second portion of the searchable file is the name of a file describing the format of the data. The format description will be described first.

A format file (created by a call to FMTBLD) contains the information necessary to describe the format of a fixed format file. That is, a list of variable names and for each variable, the type of the variable, the low and high columns, and the record number within a logical record which contains the data value.

This format, for example, could describe card input of fixed format or a disk file of fixed format created by another program or by other means. The record number required allows for multiple cards to make up one logical set of data. Thus, certain variables would always be on the first card of a pair and other variables would be on the second card of the pair. It is possible for a single column to contribute to multiple variables. For example, if the date in the form YYMMDD were in col. 1-6 then DATE in col. 1-6 could be one variable and MONTH in col. 3-4 could be another.

The data file is any fixed format ASCII file where each record (card image) is no longer than 132 characters. A fixed format file is a file where the value of a particular variable is always in the same location on a record. Thus, if a certain variable is in columns 6-15 in one record then it is always in col. 6-15 in every record.

If it is desired to use the searchable file for interactive input then the data file has the name TERM. In this case, when DBGET is called to retrieve the next values, the user will be prompted by variable name to input the values from the terminal.

The values are checked for type and an opportunity is provided for the user to verify the values. If a particular format is to be used only for interactive input then the low column, high column and record designation in the format are ignored. They must be legal values for FMTBLD, however.

The DATMAN subroutines that can be used with searchable files are DBOPN (for input only), DBNAME, DBSCAL, DBGET AND DBCLOS.

The use of the interactive input facilities provides a simple to use mechanism which yields type checking and verification options as well as prompting by variable name.

Defining Rows by Ranges

In some DATMAN calls (DBCHNG, DBDELT, DBFIND) it is possible to have the call act upon multiple rows at one time. DATMAN uses the concept of "range of values" to specify which rows are to be affected. That is, each call specifies a range of values and any row whose values are within that range is affected by the call. The ranges are specified by passing arrays of values which will be called LOWVAL (lower bound) and HGHVAL (upper bound). The use of the LOWVAL, HGHVAL arrays to define a subset of the rows can vary from finding all rows with a particular value of a variable to very general types of subsetting.

If it is desired to find the subset of the rows with a particular value of a particular variable the value of that variable would be inserted into an array using DBVINS. The variable must be an index variable. That array is now used as both LOWVAL and HGHVAL in the calling sequence of the appropriate subroutine.

If it is desired to find all rows with a single value of one variable and a single value of a second variable then both values would be inserted into an array with DBVINS. This array would then be used as both LOWVAL and HGHVAL in the calling sequence of the appropriate subroutine.

This use of the same array for both LOWVAL and HGHVAL allows what is commonly called multi-key access to a data base. By using different values for LOWVAL and HGHVAL more general types of subsets can be defined. Again, consider the case of a single variable. If it is desired to act upon all the rows with values of a single variable within a range then place the lower value of the range in LOWVAL and the upper value in HGHVAL. This will define a subset with all rows with values of that variable greater than or equal to the lower value of the range and less than or equal to the upper value of the range. Note that values equal to either bound are included in the subset.

This same range concept could be extended to values of several variables. Thus, each variable will have a lower bound and an upper bound of values and any row with all of the values of the specified values within the bounds will be included in the subset.

Section III gives several examples of the use of this range concept.

Several cautions need to be observed when using the LOWVAL and HGHVAL arrays. First, when values are inserted into an array using DBVINS, an identification code is part of that calling sequence. That identification code ties that value to the variable that was given that code when DBNAME was last called for this logical data base. This is the mechanism used throughout DATMAN to associate values and variables. The same variables should occur in both LOWVAL and HGHVAL.

Secondly, all variables used in specifying a subset of rows must be index variables. This should be kept in mind when creating the data base. In other words, specify those variables as index variables which will be used to define subsets.

Finally, the type of the variable is important when defining a range of values. If the variable is a real variable then tests for equality are unreliable in certain cases because the results of these tests depend upon the internal representation of the numbers. If the variable is alpnanumeric then the ASCII collating sequence is used. When comparing two character strings the shortest is filled to the right with blanks until the lengths of the two strings are identical. A blank is the smallest ASCII character. Then a left to right

character by character comparison is done to find the leftmost character where the two strings differ. Within this spot in the strings the character with the smallest ASCII value belongs to the smallest string. This results in B being less than BA and it also results in ? being less than 10. ASCII variables should be used in the range specification with caution.

JOINing Data Bases

The JOIN of two data bases is a logical data base (which has no physical existence) which contains all of the variables and all of the data contained in the two data bases. It can be visualized as placing the two data bases side by side. This yields a data base which has variables from both data bases and as many rows as the left hand data base. A complicating factor is how the data rows from the right hand data base are lined up with the rows from the left data base. This alignment is done by the value of the primary index variables of the right hand data base. This is perhaps best explained with an example. Figure 1 displays a data base named ANALYT. This comes from a geographical sampling experiment. A logical problem exists with the data base displayed in Figure 1. The problem is the redundancy inherent in replicating LAT and LONG for each occurrence of STATID. When doing a retrieval it is desirable to have LAT and LONG in the same data base as STATID so that the location of a data point can be determined. On the other

hand, when doing maintenance it is tedious and leads to error to change LAT for every occurrence of a particular value of STATID. The solution is to implement an operation called JOIN. This allows two physically distinct data bases (as shown in Figure 2) which for retrieval purposes are JOINed to produce the logical data base displayed in Figure 1. For maintenance purposes each occurrence of STATID corresponds to only one row in the second data base and this is the data base to which any changes are made.

The JOIN of the data base A with data base B is obtained by abstracting the primary index variables of data base B from the row of interest in data base A. These values are then used to logically attach the correspoinding row in data base B.

How best to break apart one logical data base into several physical data bases without losing or gaining any information is a subject of research into data base systems. DATMAN allows JOINs only for retrieval and requires all of the primary index variables of the JOINed data base to be present with the same names in the data base being JOINed.

The use of the JOIN allows as a special case a hierarchical view of a data base. Each level in the hierarchy has as variables the primary index variables of the next higher level. Then by JOINing the appropriate number of levels up the hierarchy all the data in a hierarchical data base is available for retrieval while still avoiding the redundant storing of data. See Figure 3 for an example of this.

Example of Data Base Analyt

SNUM	STATID	PBCONC	LAT	MGCONC	LONG
85602	ICEA	3.56	42.365		85.62
36805	ICEB		43.423		80.854
85603	ICEA	6.28	42.365		85.62
11000	2CEA	5.8	40.832		89.63
.	.	.	.		.
.	.	.	.		.
.	.	.	.		.

- SNUM is used to distinguish among rows
- STATID is used to speed up retrieval

This data base represents the sampling situation of taking samples at stations and analyzing them for metals.

Figure 1

Physical Data Bases

DATA

SNUM	STATID	PCONC	MGCONC
85602	ICEA	3.56	
36805	ICEB		
85603	ICEA	6.28	
11000	2CEA	5.8	

LOC

STATID	LAT	LONG
ICEA	42.365	85.62
2CEA	40.832	89.63
ICEB	43.423	80.854

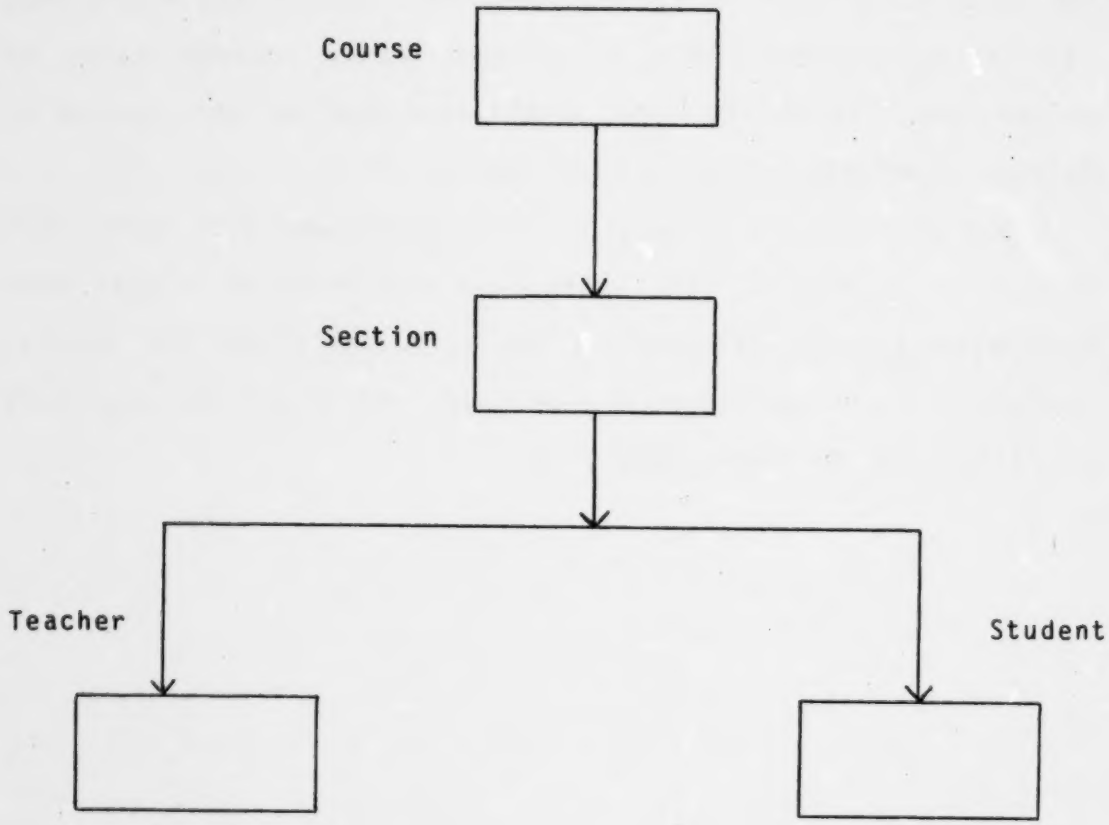
Then

Data join LOC yields

Analyt logical data base

Figure 2

Hierarchical View of Data



In DATMAN

Separate physical data base for each category above. Then to retrieve all courses taught by a teacher use TEACHER JOIN SECTION JOIN COURSE as a logical data base. JOIN allows use of distinct physical data bases in hierarchical scheme while still having logical access to one path in hierarchy.

Figure 3

Data Base of Names

When a data base is created the information about the variables for that data base is placed into a distinct "names" data base. The name of this data base is the name of the original data base followed by DD. e.g., for data base ANALYT, the names data base is called ANALYTDD. Within the names data base it is possible to add variable names, change names or delete names using the same procedures used to add, change or delete information from a normal data base.

Caution should be used when manipulating the names data base. It is possible to destroy the integrity of a data base by making certain changes. Appendix A describes the logical structure of a data base and provides the knowledge necessary to manipulate the names data base.

Data Independence

One of the virtues of a data management system is the ability to write programs that are data format independent. This notion is expanded below.

A typical program that reads input and produces output must worry about the format of the input and output.

The format is specified in terms of column position and the program makes an implicit translation between the columns of the input and the variables into which this information is read. This transformation is given by the relationship between the READ and FORMAT statements.

The problem with embedding this relationship into a program is that the program must be modified whenever this relationship changes. Thus, suppose the program was written assuming that VAR1 comes on the first card (or record) of a two card (record) pair and that VAR2 comes on the second. Then, if a set of data cards is received in which VAR2 comes on the first card and VAR1 comes on the second, the program must be modified to accommodate the new relationship between data format and the variables. If it is desired to have the program accept data in either of the two formats then more modifications must be made to the program.

Another possible solution is to write a short program that takes data in one format and converts them to the desired format.

Either of the solutions proposed requires either modifying an existing program or writing a new one. It would be preferable to have the program not make any assumptions about the data format and only retrieve values of the variables by name of the variable. The format would be specified externally to the program and hence the program would be independent of the data format.

Data independence occurs when a problem program does not need to know the format of data items but only the names of these items. Some data independence is provided by the object time format feature of FORTRAN but this feature does not have all of the flexibility that is desirable.

Data independence becomes especially desirable when a collection of interrelated programs accesses the same data. In this case, a change in the format of the data could affect all of the programs in the collection. If the programs had been written to be data independent then no modifications would have to be made.

The use of DATMAN data bases to maintain data allows a collection of interrelated programs to access a single data base in a data independent fashion.

Use of DATMAN Subroutines

Prior to any DATMAN subroutine calls the DATMAN system must be initialized. This involves a call to DBINIT which tells DATMAN which FORTRAN units are available to it.

The general approach to using a data base is to open a data base (DBOPN) by furnishing a data base name and a work array. This work array contains all of the information pertaining to this data base and is an important parameter to all of the subsequent calls. Once a data base is open then the variable names (DBNAME) are made known, the retrievals or modifications are done (DBGET, DBADD, DBDELT or DBCHNG), and the data base is closed (DBCLOS) using the work array as a parameter to the various calls.

In order to access multiple data bases a distinct work area is maintained for each data base. The work area used in the subroutine call determines which data base is accessed by each call. Examples of this are given in the Section III.

Once a data base has been opened and any JOINs accomplished then the user needs to define the names (by DBNAME) that are known for subsequent retrieval or modification. The variables that are identified are the only ones that can be accessed unless the collection of known names is redefined.

When the variable names are defined to the system each name is assigned an index by the user. This index is used to connect the variables with the appropriate values for subsequent use. For example, if we define the names VAR1, VAR2, VAR3 with indices of 1, 2, and 3 respectively then subsequent values that have an index of 1 are values of VAR1, values that have an index of 2 are values of VAR2 and values that have an index of 3 are values of VAR3.

The index that is assigned to a variable is totally at the discretion of the user. In the above example, we could just as easily define VAR3 with index 1, VAR1 with index 2, and VAR2 with index 3. The largest value of the index used is one of the factors that determines the size of the work array. It is desirable to assign the indices to variables in a sequential fashion. I.E.,1, 2, 3, etc. DBVINS and DBVEX are the subroutines used to associate values and indices.

There is no limit to the number of variables that can be known at any one time but the size of the work array depends upon the number of variables that are known at any time as well as other parameters. See the section Calculating Array Dimensions for a discussion of this.

If multiple data bases are open then the work area used in the call making the names known controls which data base the variables are in. Thus, there is no ambiguity in using VAR1 simultaneously in two different data bases since the values are inserted and extracted based on the work area of the appropriate data base.

This reliance on the names of the variables rather than requiring any knowledge of the physical location of the variable within the data base is the notion of data independence. This is one of the features of data base systems which makes such systems useful for such a wide variety of applications. The user need not be aware of the physical structure of the data base at all and is unconcerned with the physical relationship of variables. The data base system, in this case DATMAN, takes care of all necessary physical formatting details. See the section on Data Independence for a further discussion.

Modification of a Data Base

Once a data base has been opened and the names to be used are made known then all preparations have been made for either modification or retrieval of data from the data base. Modification is dealt with first because there are fewer options involved. Before either is discussed, however, the techniques for passing values of variables is discussed.

Whenever accessing a DATMAN data base using the FORTRAN interface it is necessary to pass the values of the variables to and from the calling program. The mechanism used in DATMAN to do this is to save all of the values in an array and pass all of the values at once. This array must be initialized using a subroutine call prior to placing any values into it.

When values are inserted into the array, the index of the variable when the names were made known to the data base and the value and length of the variable must be specified. When values are extracted from the array then the number of the variable and the array are furnished and the value and length are returned. Undefined values are designated by a length of 0. DBVINS and DBVEX are the insertion and extraction subroutines used to pack and unpack values into the value array.

Three ways exist to modify an existing data base. New rows can be added, data in existing rows can be changed, or existing rows can be deleted. In all of these operatons those variables which have been identified as primary index variables play an important role.

Primary index variables are those variables whose values are used to identify and distinguish among rows. Because of this use to identify a particular row, any two rows must have values of the primary indices which differ in at least one component.

When the add subroutine is called one row is added to the data base. The only information that is necessary to supply is the values of the variables which are to be defined in the new row. Any variables whose value is not given remains undefined. All components of the primary index must be assigned

values for any new row and DATMAN will return an error code if there is already a row with the same values of the primary indices. This prohibits duplication of the row identification and guarantees that it be possible to access an individual row. The subroutine DBADD is used to add data to a data base.

When adding data to a data base only one row is added at a time. When changing data or deleting rows it is possible to affect multiple rows with one call. This is accomplished by specifying a range of values for the primary indices. Any row whose values of the primary indices are all within the range is either changed or deleted, as appropriate. If only one row is to be changed then use the value array which defines this row as both the lower bound and upper bound in the calling sequence. See section entitled Defining Rows by Ranges.

If multiple rows are to be modified then confusion is possible when either multiple primary index variables exist or when a primary index variable is alphabetic. Alphabetic index variables cause problems because of the fact that all embedded blanks are deleted and they are considered left justified in the field. Comparisons between two alphabetic variables are made using the ASCII collating sequence and this causes A and AA to precede B. It also causes 10 to precede both 20 and 2. Thus, use of alphabetic primary index variables to define a collection of rows should be with awareness of how numbers - when considered as characters - are sorted.

Confusion caused by multiple primary index variables is possible because of the effect of omitting one or more of the primary index variables. In this case, only those variables whose values are furnished are used to define the rows affected. Any value of the omitted variables is acceptable. As an example of the use of multiple primary index variables to define the subset or rows of interest suppose V1 and V2 are both primary index variables and are both integers. Then if the rows are specified with the lower bound of V1 as 10 and the lower bound of V2 as 20 and the upper bound of V1 as 15 and the upper bound of V2 as 30 then those rows with $10 \leq V1 \leq 15$ and $20 \leq V2 \leq 30$ will be affected. If, on the other hand, the row specification is a lower bound of V2 of 20 and an upper bound of V2 of 30 and V1 omitted than all rows with $20 \leq V2 \leq 30$ will be affected regardless of the value of V1 for any of those rows.

In the delete subroutine (DBDELT) the row definition needs to be furnished. The rows that are defined in terms of the primary index variables specified are deleted as well as all values of any variables in that row. The row no longer exists logically. This is different than having the values of the variables for that row undefined.

In the change subroutine (DBCHNG) other information must also be furnished. This includes primarily the list of new values of variables for the rows that are affected. This can include values of the primary index variables as well as non-index variables. The appropriate rows are located using old values of the primary variables and then the changes are applied. On subsequent calls, the new values of the primary index variables are used to define the row. Any changes to primary index variables must not result in the duplication of an existing collection of values.

The other information that must be furnished the change subroutine are the values of the variables that are to be used to verify the existing values. None of these need be furnished. A defined row is tested to see if all of the values of the verification variables are the current values of these variables. If any value is not the current value then the row is not modified.

Finally, whenever a data base has been opened for output (which is necessary to allow modification) the data base must be closed. If the data base is not closed prior to terminating the program then necessary pointers maintained within the data base may become incorrect. If this happens execution of the command INDEX within the terminal interface will result in a consistent data base although some changes may not have been made.

Retrieval of Data from Data Base

The basic mechanism for the use of the retrieval subroutine is that a series of calls (possibly none) are made defining the subset of rows that are desired and the conditions of their retrieval. Once the context of the retrieval has been defined then the routine DBGET is called to retrieve the rows in sequence. It returns the values of the variables known to the data base for the retrieved row. When the defined subset is exhausted then an end of file flag is returned.

There are four separate calls that can be used to define the context of the retrieval. These are:

1) DBJOIN which defines a logical data base to be the logical merger of an open data base and a second named data base.
2) DBFIND which defines a subset of the rows in terms of the values of index variables.
3) DBORDR which allows the rows to be returned ordered according to the values of a particular variable.
4) DBSCAL which specifies the return of a certain percentage of the rows. Every one of two for example. This is useful in graphics applications.

These calls are all independent of each other and some or all can be made. If none of the context defining calls are made the rows are returned in physical order. The considerations of two data bases for a discussion of the JOIN

involved in a JOIN will be discussed first. See the section on JOINing. The JOIN operation constructs a logical data base from several physical data bases. In JOINing data bases, it is important to distinguish between the data base being joined (the second of the two data bases) and the data base being joined to (the first of the two data bases). For the remainder of the discussion on JOIN, DB1 JOIN DB2 will be the assumed two data bases.

Prior to calling DBJOIN, DB1 must be opened. The call to DBJOIN with the work area for DB1 and the name DB2 defines a logical data base. Following the call to DBJOIN the names are defined by a call to DBNAME. The names defined by the call to DBNAME are the names in the logical data base defined by the JOIN, any duplicate names are taken from the initial data base.

When calling DBJOIN we have the following constraints to be considered:

1) DB1 can be either a DATMAN data base or a logical data base defined from previous call to DBJOIN.
2) DB2 must be a DATMAN data base.
3) The primary index variables of DB2 must be defined, using the same names, in DB1.

Once the data base has been opened (and the logical data base defined through JOINs if desired) the next step is to make known the names using DBNAME. This is as previously discussed. Once the names have been defined then a context for

the retrieval is set and the retrieval itself can be accomplished. If no context is established then the retrieval is done sequentially through the data base. The rows are returned in physical order. The retrieval of the rows is accomplished using the DBGET subroutine.

The subroutine DBGET is called to retrieve the next row in whatever context is desired. It returns the values for the variables known to the user packed into an array. The utility subroutine DBVEX is called to extract the value from the array in which it is returned by DBGET and move it into a variable specified by the user. Since the values can be integer, real or alphanumeric, the user should be aware of the type of the variable before extracting it from the value array. This allows movement of the value into a variable of the appropriate type. If the program wishes to dynamically determine the type of the variable then the utility function DBTYPE can be used.

If the user wishes to retrieve a subset of the rows of a data base then the subroutine DBFIND is used to define the desired subset. The subset is defined by the values of the variables in the rows and DBFIND is called with a lower bound for that set of values and an upper bound for that set of values. If only one row is desired then the lower bound and the upper bound can be identical. These bounds can be defined in terms of more than one variable. In this case the rows that are selected are the rows where the value of the first variable

is within the bounds for the first variable and the value of the second variable within the row is within the bounds for the second variable and so forth. See Section III for examples of this. All variables that are used in DBFIND to define the subset must be index variables. If an alphanumeric variable is used to do the row definition then the values are ordered alphanumerically not numerically. That is, if the variable has values 10 and 2 then 10 precedes 2 in the alphanumeric ordering. See section entitled Defining Rows by Ranges for further discussion.

When calling DBFIND three possibilities exist for dealing with the subset of rows defined. These are:

1) The new rows can be a new definition of a subset.
2) The new rows can be merged (ORed) together with the rows previously defined. A row is in the result if it is either the current definition or in a past definition.
3) The new rows can be intersected (ANDed) with the rows previously defined. A row is in the result if it is in both the current definition and in a past definition.

The normal order of retrieval is dependent upon the physical order of the rows in the data base. It is possible to retrieve the rows in ascending order of the values of a particular variable. The only restriction upon this ordering is that the variable must be an index variable. The subroutine

that sets the order is DBORDR. A normal retrieval returns all rows which meet the conditions of the retrieval. In certain applications (most notably graphics) not all rows would be desired. It is possible to retrieve a systematic subset of the rows defined by calling the subroutine DBSCAL with an increment. This increment is a real number which represents the fraction of the rows desired. Thus, if 1 of 3 rows are desired the real number .333333 would be the increment. If 2 of 5 rows are desired the real number .4 would be the increment.

The subroutine DBFIND, DBORDR, and DBSCAL are all optional and must be called after DBNAME and before any retrieval from the data base. Once retrieval has begun, redefining the context of that retrieval will terminate the original retrieval sequence and start again with the new retrieval context.

All of the subroutines discussed have an array called the work array as a parameter. This work array contains all of the information specific to the status of a particular open data base. It is possible to have different data bases open simultaneously by having distinct work arrays for each data base. The action upon one data base is independent of the action upon another. Thus, for example, it is possible to use data retrieved from one data base to update another without having to re-open or close either of the data bases.

SECTION II - REFERENCE

Dimension of DATMAN Arrays

DATMAN uses two different types of arrays within the FORTRAN calling sequences. These are the work array which controls the logical data base being accessed and the value array which is used to pass values to and from the subroutines. Both of these array types must be dimensioned to a size adequate to hold the information DATMAN places within them. The array types will be discussed separately but all word sizes used in calculating the necessary bounds are INTEGER*2 (16 bit) words.

First, the value array types are discussed. The dimension of this array depends upon the number of values and the type of the values placed within it. Integer values take up four words, real values take up six words and character values take up [n/2]+2 where n is the number of characters in the value and the symbol [ ] means round up if n is odd. In addition one word is used for bookkeeping purposes. The minimum array size is found by adding one to the sum of the above lengths for each value of each type. More formally, if M values are to be placed in the array and

$$l_i = \begin{cases} 4 & \text{if value}_i \text{ an integer} \\ 6 & \text{if value}_i \text{ is a real} \\ [n/2] + 2 & \text{if value}_i \text{ is a character string of length } n. \end{cases}$$

$$\text{then minimum dimension} = 1+\sum_{i=1}^{M} l_i$$

The size of the second array type, the work array, depends upon the use of the data base. The size of the array depends upon:

1) the number of index variables in the data base
2) the number of variables known to the data base and
3) whether another data base is to be joined to this data base.

Suppose a single data base is being accessed. Then the minimum dimension of the work array is (in integer*2 words)

50 + 50* number of index variables in data base
+ 3* number of variables made known in DBNAME

If another data base is to JOINed to the one controlled by the work area then to the above must be added

1 + 3* number of primary index variables for data base to be JOINed.

In this case (a JOIN) the work area must also be large enough to include the work area for the subsequent data bases.

If the data bases DB_1, DB_2, ... , DB_N are to be JOINed in that order then the minimum array size for the work area controlling the resulting logical data base can be found by the following formula:

N + 50N + 50* (total number of index variables in all N data bases)
+ 3* (total number of primary variables in data bases 2,...,N)
+ 3* number of variables made known in DBNAME

It can be observed from these formulae that the number of index variables in a data base has a direct impact on amount of memory used and this should be taken into account when selecting the index variables.

Basic DATMAN Calls

DBINIT (LOW)

DBINIT is a subroutine that is called once to initialize the DATMAN system. DATMAN needs to know which FORTRAN logical units are available to it. It will use only FORTRAN units with number LOW or above and it will assume that all FORTRAN units numbered LOW or greater are available to it.

The arguments to DBINIT

LOW - INTEGER*2 which gives the lower bound of the FORTRAN units available to DATMAN

DBOPN (DBNAME, LENG, WORK, RWKEY, ERROR)

DBOPN is a subroutine which opens (or makes available) a data base for use. It is also possible to open a searchable file for retrieval. DBNAME is a character string which gives the name of the data base. It has LENG characters. WORK is an array which contains all of the information relevant to the data base. This array will be passed on to all subsequent calls which use this data base. RWKEY is a key which tells DATMAN wnether the data base is to be opened for reading or writing. Searchable files can be opened only for reading. ERROR is a return code which is 0 if open was successful and non-zero if unsuccessful.

The arguments to DBOPN are:

DBNAME - The name of the data base to be opened in characters.

LENG - INTEGER*2 length of DBNAME in characters.

WORK - INTEGER*2 array which controls this data base.

RWKEY - INTEGER*2

1 - if open for input

2 - if open for output

ERROR - INTEGER*2

0 - if open successful

1 - if DBNAME is not valid data base or searchable file name

2 - if attempting to open non-data base searchable file for output

DBNAME(WORK, VARNAM, ERROR)

DBNAME is a subroutine that is called to make known to the user a set of names. WORK is the same array that was used in DBOPN and contains the information to identify which data base is being referenced. VARNAM is an array which contains the list of names which the user wishes to manipulate. These names are in DATMAN value format (using DBVINS). ERROR is an error code indicator.

When a name is placed in VARNAM with DBVINS an identification code is assigned to the name. This code is used to identify the values of that variable for all subsequent calls to DATMAN subroutines. The identification codes should be assigned beginning with 1. That is, if ten names are to be made known to the user the codes 1-10 should be used. Once a variable has been associated with a code that association remains in force until either the data base is closed (DBCLOS) or another call to DBNAME is made.

If the data base being accessed is the JOIN of several data bases then the names are defined in terms of the resultant logical data base. If the same variable name occurs several times in the logical data base, the first occurrence is the one for which values are returned.

If a second call to DBNAME occurs for a particular data base then it has the same effect as closing the data base, reopening it and calling DBNAME again. That is, any retrieval context (set by calls to DBFIND, DBORDR, DBSCAL) is lost and the variables previously known to the user are no longer available (unless included in second DBNAME call).

The arguments to DBNAME are:

WORK - INTEGER*2 array which was used with DBOPN and which controls data base

VARNAM - INTEGER*2 array used to define names to be known to user. Names placed in VARNAM by DBVINS.

ERROR - INTEGER*2 - error return

0 - if no errors

1 - if invalid work area (either not opened correctly or modified by user)

2 - if some of the names are not in data base. Use DBLENG to determine which names are undefined.

DBCLOS (WORK)

DBCLOS is called to logically disconnect a user from a data base. It is important to close the data base before terminating the program. If the program terminates without closing the data base and the data base was opened for output then the integrity of the data base may be impaired.

The arguments are:

WORK - INTEGER*2 array which was used with DBOPN and which controls data base.

Modifications to Existing Data Base

DBADD (WORK, VALUE, ERROR)

DBADD is the subroutine that adds a new row to an open data base. The data base must have been opened for output. WORK is the array that controls the data base. VALUE is the set of values to be entered into the newly created row. Values are placed into the array VALUE using DBVINS and only those variables that have values need have them entered. All variables for which values have not been assigned will be maintained as undefined. Values must be entered for all primary index variables.

The arguments to DBADD are:

WORK - INTEGER*2 array which was used with DBOPN and which controls data base

VALUE - INTEGER*2 array in DATMAN value format which contains values to be added to new row. Values placed into VALUE using DBVINS.

ERROR - INTEGER*2 return code

- 0 - new row added
- 1 - if invalid work area (not open correctly or modified by user or opened for input)
- 2 - undefined value of primary index variable
- 3 - attempt to duplicate values of primary index variable.

DBCHNG (WORK, LOWVAL, HGHVAL, VERVAL, NEWVAL, NUMCHG, NUMNVR, NUMDUP, ERROR)

DBCHNG is the subroutine which modifies one or more existing rows in a data base. The rows to be modified are defined through the arrays LOWVAL and HGHVAL which could bracket multiple rows. It is possible to test each row for existing values of variables and that is accomplished with the array VERVAL. The new values for the variables are passed in the array NEWVAL. If a value is to be made undefined it should be entered with a length of 0. Returned are the number of rows successfully changed, the number of rows which were not changed because the verification of the row failed and the number of rows which were not changed because applying the changes would have resulted in duplicate values of the primary index variable.

The data base must have been opened for output and all value arrays are in DATMAN value format constructed using DBVINS. The variables used to define the rows (in LOWVAL and HGHVAL) must be index variables for the data base (although not necessarily primary index variables). Any non index variables in LOWVAL and HGHVAL will be ignored.

The verification procedure compares, for each variable to be verified, the existing value in each row selected with the values in the array VERVAL. If no verification is desired then this argument should be a 0.

The rows to be modified are specified by the arrays LOWVAL and HGHVAL. Each array is a list of values of index variables (any non index variables are ignored). Any variable that occurs in LOWVAL must also occur in HGHVAL. Any row which has values of each variable in the list that is greater than or equal to the corresponding value in LOWVAL and less than or equal to the corresponding value in HGHVAL is chosen for modification. One particular row could be specified by making LOWVAL and HGHVAL the same values and using only primary index values. If one of the variables is alphanumeric then care should be used when doing tests for greater than or less than. For alphanumeric variables the value 10 is less than the value 2 and this could cause problems if alphanumeric ordering is not well understood.

The arguments to DBCHNG are:

WORK - INTEGER*2 array which is used by DBOPN and which controls the data base.

LOWVAL - INTEGER*2 array in DATMAN value format which gives the lower bound for the rows to be modified. Constructed using DBVINS.

HGHVAL - INTEGER*2 array in DATMAN value format which gives the upper bound for the rows to be modified. Could be identical to LOWVAL and must contain the same variables in list. Constructed using DBVINS.

VERVAL - INTEGER*2 array in DATMAN value format which gives the values which are used for verification. No modifications are made to a row if all of the values in VERVAL are not identical to the corresponding values in the row. Constructed using DBVINS. 0 if no verification desired.

NEWVAL - INTEGER*2 array in DATMAN value format which gives the new values for the specified variables. Constructed using DBVINS.

NUMCHG - INTEGER*2. Returned by DBCHNG, NUMCHG is the number of rows to which changes were applied.

NUMNVR - INTEGER*2. Returned by DBCHNG, NUMNVR is the number of rows to which changes were not applied because the row failed verification.

NUMDUP - INTEGER*2. Returned by DBCHNG, NUMDUP is the number of rows to which changes were not applied because to do so would result in duplication of existing primary index values.

ERROR - INTEGER*2. Returned by DBCHNG.

0 - no errors

1 - Work area incorrect. Either not properly opened or modified by user or opened for input.

2 - LOWVAL and HGHVAL contained no index variables or variable used in LOWVAL which did not occur in HGHVAL.

DBDELT (WORK, LOWVAL, HGHVAL, NUMDEL, ERROR)

DBELT is the subroutine which is called to delete rows from an existing data base. The number of rows deleted can be from one to the total data base depending on the specifications given in LOWVAL and HGHVAL. If a row is deleted all the data that was in any variable on the row is gone. If it is desired to delete certain variables from the row then DBCHNG should be used.

WORK is the array which controls the particular data base which is to have rows deleted. LOWVAL and HGHVAL are the arrays which specify the rows to be deleted. Both must contain the same variables in the list and only the index variables in the arrays will have any effect. Any row that has values of each variable in the list that is greater than or equal to the corresponding value in LOWVAL and less than or equal to the corresponding value in HGHVAL is deleted. If one of the variables is alphanumeric then care should be used when specifying tests for greater than or less than. For alphanumeric variables the value 10 is less than the value 2 and this could cause problems if alphanumeric ordering is not well understood.

The arguments to DBELT are:

WORK - INTEGER*2 array which is used by DBOPN and which controls the data base.

LOWVAL - INTEGER*2 array in DATMAN value format which gives the lower bound for the row to be modified. Constructed using DBVINS.

HGHVAL - INTEGER*2 array in DATMAN value format which gives the upper bound for the rows to be modified. Could be identical to LOWVAL and must contain the same variables in list. Constructed using DBVINS.

NUMDEL - INTEGER*2. Returned by DBELT and gives the number of rows actually deleted.

ERROR - INTEGER*2. Returned by DBELT with values.

0 - No errors

1 - Incorrect work array. Either improperly opened or modified by user or opened for input.

2 - None of the variables in LOWVAL or HGHVAL was an index variable or variable in LOWVAL that was not in HGHVAL.

Retrieval Calls to DATMAN

DBFIND (WORK, LOWVAL, HGHVAL, FLAG, ERROR)

DBFIND is the call that defines a subset for retrieval. Within the subset defined, rows are retrieved one at a time with DBGET. If no call to DBFIND is made all of the rows are retrieved. The subset that is defined with this call is specified with the arrays LOWVAL and HGHVAL. WORK is the array that controls the logical data base from which retrieval is to be done.

Any call to DBFIND will either be merged with the subset defined on a prior call to DBFIND (for the same logical data base) or will supercede the prior subset. This is controlled by the parameter FLAG. The three possible options are:

1) define a new subset and supercede any prior subset,
2) include a row in the current subset only if it was included in the prior subset and the values of the variables in the row are in the range defined by the current arrays LOWVAL and HGHVAL,
3) include a row in the current subset if it was included in the prior subset or the values in tne row are in the range defined by the current arrays LOWVAL and HGHVAL.

Calls to DBFIND are valid whether the logical data base defined by the WORK array is a single physical data base or is several physical data bases JOINed together. It is not valid

when WORK defines a searchable file which is not a data base. If WORK controls more than one physical data base, the variables used in the LOWVAL and HGHVAL arrays must be index variables in the first of the physical data bases defined by WORK. That is, the data base that was opened, not the ones that were JOINed. The use of the LOWVAL, HGHVAL arrays to define a subset of the rows can vary from finding all rows with particular value of a particular variable to finding all rows with different variables in different ranges.

If it is desired to find all rows with a single value of a variable then the value of that variable would be inserted into an array using DBVINS. The variable must be an index variable. That array is now used as both LOWVAL and HGHVAL in the calling sequence.

If it is desired to find all rows with a single value of one variable and a single value of a second variable then both values would be inserted into an array with DBVINS. This array would then be used as both LOWVAL and HGHVAL in the calling sequence.

This use of the same array for both LOWVAL and HGHVAL allows what is commonly called multi-key access to a data base. By using different values for LOWVAL and HGHVAL more general types of subsets can be defined. Again, consider the case of a single variable. If it is desired to find all the rows with values of a single variable within a range then place the lower value of the range in LOWVAL and the upper value in HGHVAL. This will define a subset with all rows with values of that variable greater than or equal to the lower value of the

range and less than or equal to the upper value of the range. Note that values equal to either bound are included in the subset.

This same range concept could be extended to values of several variables. Thus, each variable will have a lower bound and an upper bound of values and any row with all of the values of the specified values within the bounds will be included in the subset. If a variable in LOWVAL is undefined (length of 0) then the corresponding value in HGHVAL is taken to be the upper bound. Similarly, for undefined values in HGHVAL.

The appendix gives several examples of the use of DBFIND.

Several cautions need to be observed when using the LOWVAL and HGHVAL arrays. First, when values are inserted into an array using DBVINS, an identification code is part of that calling sequence. That identification code ties that value to the variable that was given that code when DBNAME was last called for this logical data base. This is the mechanism used throughout DATMAN to associate values and variables. The same variables should occur in both LOWVAL and HGHVAL.

Secondly, all variables used in specifying a subset of rows must be index variables. This should be kept in mind when creating the data base. In other words, specify those variables as index variables which will be used to define subsets using DBFIND. Any non index variables in LOWVAL or HGHVAL will be ignored.

Finally, the type of the variable is important when defining a range of values. If the variable is a real variable

then tests for equality are unreliable in certain cases because the results of these tests depend upon the internal representation of the numbers. If the variable is alphanumeric then the ASCII collating sequence is used. When comparing two character strings the shortest is filled to the right with blanks until the lengths of the two strings are identical. A blank is the smallest ASCII character. Then a left to right character by character comparison is done to find the leftmost character where the two strings differ. Within this spot in the strings the character with the smallest ASCII value belongs to the smallest string. This results in B being less than BA and it also results in 2 being less than 10. ASCII variables should be used in the range specification with caution.

The arguments to DBFIND are:

WORK - INTEGER*2 array which controls this logical data base.

LOWVAL - INTEGER*2 array which gives the lower bound for the subset defined. Created using DBVINS.

HGHVAL - INTEGER*2 array which gives the upper bound for the subset defined. Created using DBVINS.

FLAG - INTEGER*2 - Determines how the subset being defined should be merged with the last defined subset.

1 - create new subset and disregard prior subset.

2 - include row only if it is in both subset specified with LOWVAL and HGHVAL and in prior subset.

3 - include row if it is either in subset specified with LOWVAL and HGHVAL or it is in prior subset.

ERROR - INTEGER*2 - Returned by DBFIND.

0 - No error

1 - Invalid work array. Either not properly opened, modified by user, or describes searchable file.

2 - No variable in LOWVAL and HGHVAL is an index variable for first physical data base in logical data base described by WORK or same variables are not used in LOWVAL and HGHVAL.

DBGET(WORK, VALUE, FLAG)

DBGET retrieves the next record in the context defined by DBFIND, DBORDR and DBSCAL. If a context has not been defined it retrieves the total data base one row at a time in physical order. If a subset has been defined it retrieves that subset one row at a time in physical order. If DBORDR has been called the retrieval is in ascending order of the specified variable and if DBSCAL has been called a systematic subset of the rows is returned. In any case, only one row is returned by each call to DBGET.

When a call to DBGET is made the next row in the context is retrieved and the values of the variables known to the user are returned in the array VALUE. These values are retrieved from VALUE by DBVEX.

The arguments to DBGET are:

WORK - INTEGER*2 array which controls the logical data base being accessed.

VALUE - INTEGER*2 array in which values of variables are returned. Values are retrieved from VALUE by DBVEX.

FLAG - INTEGER*2 - Returned by DBGET giving status of retrieval.

0 - Next row successfully retrieved.

1 - End of file on subset of rows in defined context.

2 - Error in WORK array. May be due to incorrectly opened data base or user modification of WORK array.

3 - Error in type of one of the variables (possible if DBGET is being used on a searchable file).

DBJOIN (WORK, DBNAM, LENG, ERROR)

DBJOIN is the subroutine that creates a logical data base from several physical data bases for the purposes of retrieval. The resulting data base has two components. The "owner" component is the logical data base controlled by WORK prior to the call and the "joined" component is the physical data base whose name is in DBNAM with length LENG. After the call the resulting logical data base will be controlled by the WORK array.

The resulting logical data base has all of the variables from both the owner component and the joined component. A data row in the resulting data base is formed by adding to the data row of the owner component the corresponding row of the joined component. The correspondence is made by matching the values of the primary index variables of the joined component. This requires that those variables also be in the owner component. This match is done by the names of the variables.

The joined component must be a physical data base and not a searchable file. The owner component can be a physical data base or a logical data base created by a prior call to DBJOIN.

The owner component must have been opened for input, DBJOIN is not allowed for any data base opened for output. The primary index variables of the joined component must be available, with the same names in the owner segment. Variables are made known through a subsequent call to DBNAME after a call to DBJOIN and any variable name which is multiply defined in the logical data base will be chosen to be the first occurrence of that name in the physical data bases that comprise the logical data base. The primary index variables of the joined component need not be among the variables being defined in the DBNAME call. Only those variables the user wishes to access need be defined in the DBNAME call.

The arguments to DBJOIN are:

WORK - INTEGER*2 array which controls the owner component prior to the call and will control the logical data base subsequent to the call.

DBNAM - character string which gives the name of the physical data base to be joined.

LENG - INTEGER*2 - Length in characters of DBNAM.

ERROR - INTEGER*2 - Returned by DBJOIN with error indicator.

0 - No errors.

1 - WORK is invalid. May be caused by not having properly opened WORK for input or by modifying WORK.

2 - DBNAM is not valid physical data base.

3 - The primary index variables of DBNAM are not available in the logical data base controlled by WORK.

DBORDR (WORK, VARNAM, ERROR)

DBORDR is called to specify the order that rows are to be returned by DBGET. If no call to DBORDR is made then the subset determined by DBFIND (or the whole data base if no call to DBFIND is made) is retrieved in an arbitrary order. With a call to DBORDR the rows are retrieved in ascending order of the specified variable. The variable used to specify the order of retrieval must be an index variable and is identified by the number used to specify the variable name to DBNAME.

The arguments to DBORDR are:

WORK - INTEGER*2 array which controls the logical data base.

VARNUM - INTEGER*2 - identification number of the variable which determines the order of retrieval.

ERROR - INTEGER*2 - Returned by DBORDR.

0 - No error.

1 - Work area invalid. Either work area improperly opened or modified by user.

2 - VARNUM is not an index variable.

DBREWD (WORK)

This subroutine rewinds the data base opened to WORK. That is, it undefines any retrieval context and undefines any variable names. The result is if the data base had just been opened and any JOINs performed but no calls to DBFIND, DBNAME, DBGET, DBORDR or DBSCAL have been made.

The argument to DBREWD is:

WORK - INTEGER*2 array which controls logical data base.

DBSCAL (WORK, FACTOR, ERROR)

The normal retrieval will return every row that has been defined by DBFIND. For certain applications (primarily graphical) it may be desirable to return a systematic subset of the rows defined. For example, return every other row because the resolution of the output device is less than the resolution of the input device which generated the data values.

FACTOR is a real number which gives the ratio between the number of rows desired to be retrieved and the number of rows in the subset defined. This makes FACTOR greater than 0 and less than or equal to 1. If every other row is desired FACTOR is .5. If two out of three rows are desired FACTOR is .666666.

The arguments to DBSCAL are:

WORK - INTEGER*2 array which controls logical data base.

FACTOR - REAL*4 number greater than 0 and less than or equal to 1 which gives the ratio between the number of rows to be returned and the total number defined in the subset.

ERROR - 0 - No error.

1 - Work area is invalid. Either improperly opened or modified by user.

2 - FACTOR is outside range 0 to 1.

DATMAN Utility Calls

J = DBINDX (WORK,I)

DBINDX is an INTEGER*2 function which returns the index code of the variable assigned the number I through DBNAME. The returned values are:

0 - not an index variable

1 - a primary index variable (used for data row identification)

2 - a secondary index variable (used for retrieval but not for data row identification)

The arguments to DBINDX are:

WORK - INTEGER*2 array which controls the logical data base.

I - INTEGER*2 number of the variable of which the index code is returned.

J = DBLENG (WORK,I)

DBLENG is an INTEGER*2 function which returns the length (in characters) of the variable assigned the number I tnrough DBNAME. If the Ith variable was not a valid name for the opened data base then DBLENG will return 0. This is the mechanism to determine which variable names were incorrect in the event of an error exit from DBNAME.

The arguments to DBLENG are:

WORK - INTEGER*2 array which controls the logical data base.

I - INTEGER*2 number of the variable of which the length is returned.

J = DBTYPE (WORK,I)

DBTYPE is an INTEGER*2 function which returns the type of the Ith variable in the logical data base controlled by WORK. The function will return

0 - error

1 - type of Ith variable is integer (32 bits)

2 - type of Ith variable is alphanumeric

3 - type of Ith variable is real (64 bits)

The arguments of DBTYPE are:

WORK - INTEGER*2 array. The array which controls the logical data base.

I - INTEGER*2 - The number of the variable for which the type is returned.

DBVEX (VALUE, I, ARG, LENG)

DBVEX is the subroutine which extracts values of particular variables from the array VALUE in which they are packed. The value of the Ith variable is returned into the variable ARG. The type of ARG depends upon the type of the variable which is being retrieved. If the Ith variable is real then ARG should be REAL*8 and LENG will be returned as 8. If the Ith variable is integer then ARG should be INTEGER*4 and LENG will be returned as 4. If the Ith variable is alphanumeric then ARG should be an array of sufficient size to hold all of the characters returned. LENG will be the number of characters returned. LENG will be 0 if the value of variable I is undefined.

DBVEX is typically used after a call to DBGET since DBGET returns the values packed into an array VALUE. If the type or length are to be determined dynamically then DBLENG and DBTYPE are to be used.

The arguments to DBVEX are:

VALUE - INTEGER*2 array from which the values are to be returned.

I - INTEGER*2 - number of the variable for which the value is to be returned.

ARG - arbitrary type - location into which the value is to be returned. Integer values in DATMAN are 32 bits and real numbers are 64 bits. Characters values are returned in ASCII format.

LENG - INTEGER*2 - length in bytes of variable whose value is being returned. 0 if variable is undefined.

DBVINS (VALUE, I, ARG, LENG)

DBVINS is the subroutine used to pack multiple values into an array. Many of the DATMAN calls use information packed into an array by DBVINS.

The array VALUE can be viewed as a depository of values. Before any values can be inserted it must be initialized (DBVINT). Each value that is inserted is identified with the value of I that is passed along with the value to be inserted in ARG. LENG is the length of the argument to be inserted into VALUE.

Since each value is identified by the corresponding value of I, for each value of I there can be only one value inserted into the array. The typical method of inserting values is to initialize the array, insert the values for the desired values of I, use the array in a call, and then reinitialize the array and insert other values for another call.

The arguments to DBVINS are:

VALUE - INTEGER*2 array - The array into which values are packed.

I - INTEGER*2 - The identification code for the particular value to be inserted.

ARG - varied type - The particular value to be inserted. Could be REAL*8, INTEGER*4 or character.

LENG - INTEGER*2 - The length of the value to be inserted. LENG should be given the correct value regardless of the type of the argument.

DBVINT (VALUE)

This subroutine is used to initialize an array prior to any insertions by DBVINS. This clears the array and must be called prior to inserting a logical collection of values into an array.

The arguments to DBVINT are:

VALUE - INTEGER*2 array. The array to be initialized.

Creating DATMAN Structures

DBBLD (DBNAM, LENG, VARNAM, TYPE, INDCOD, VARLEN, RECSIZ, RESERV, ERROR)

DBBLD is the subroutine which creates a DATMAN data base with no data in it. In order to create a data base the name of the data base must be furnished as a character string (DBNAM) with length LENG, the list of variable names must be furnished and for each variable name, the type, the length (if alphanumeric) and the index code of the variable must be furnished. Also needed is the size of a single row in words.

The names are inserted into the array VARNAM by DBVINS and each name is assigned a code with that insertion. This code is used as an index into the arrays TYPE, INDCOD, and VARLEN so that DBBLD can associate the appropriate information with each variable.

DATMAN needs to know the maximum length of a single row. This is at least 25 words for system needs plus the sum of the lengths of the variables (two words for integers, four words for real numbers and the number of characters assigned for alphanumeric variables). It is also possible to reserve additional space in each record for the addition of additional variables. This space must be reserved if any additional variables are to be added to the list of variables or if certain types of modifications are to be made to existing variables after the data base is created.

The arguments to DBBLD are:

DBNAM - character - The name of the data base to be constructed. The name is the initial segment of the name of certain files constructed by DATMAN and so it should be long enough to be meaningful but short enough to conform to system constraints. (The PDP11/70 allows a maximum of 9 characters in the initial segment.)

LENG - INTEGER*2. Length of DBNAM in characters.

VARNAM - INTEGER*2 array. The list of variable names for the data base. Each name is assigned a unique integer code from one to the number of names which is used as an index into TYPE, VARLEN, INDCOD arrays. A valid DATMAN variable name is at most ten characters with only letters, numbers, $ and _ allowed as characters. This array is constructed using DBVINS.

TYPE - INTEGER*2 array. Used to pass the type of the variable list. The Ith entry in the array is associated with the name given the code of I in VARNAM. Values are:

1 - Integer (32 bits)

2 - Alphanumeric (a maximum of 15 characters for index variables and a maximum of 80 characters for other variables).

3 - Real (64 bits)

INDCOD - INTEGER*2 array. Used to pass the index code of the variable list. The Ith entry in the list is associated with the name given the code of I in VARNAM.

Values are:

0 - Not an index variable.

1 - Primary index variable. One of the components of the row identification variables. At most 7 primary index variables are allowed.

2 - Secondary index variable. Not a primary index variable but a variable which will be used to subset the rows for retrieval.

VARLEN - INTEGER*2 array. Used to pass the length in characters of the data items of the variables. The Ith entry in the list is associated with the name given the code of I in VARNAM. The length is only meaningful for alphanumeric variables. These have a maximum length of 15 characters for index variables and 80 characters for non-index variables.

RECSIZ - INTEGER*2. Length in words of a row of data for this data base. Must include 25 words for system requirements.

RESERV - INTEGER*2. Must be a 0. This space is reserved for future applications requiring journaling (audit trails).

ERROR - INTEGER*2. Error code returned by DBBLD

0 - data base created

1 - invalid data base name

2 - invalid variable name

3 - invalid type

4 - invalid index codes

5 - invalid length

6 - invalid record size

FMTBLD (FMTFIL, LENG, VARNAM, PHYREC, LOWCOL, HGHCOL, TYPE, TOTREC, ERROR)

FMTBLD is the subroutine which creates a DATMAN format file. A DATMAN format file is used to describe a fixed format ASCII collection of records. An example might be a card image file with a fixed format on each card. Use of a DATMAN format allows describing each field by a variable name. A format file is a component of a searchable file. A searchable file (created by SFBLD) can be used for retrieval by DBGET. This enables a programmer to take advantage of the data independence features of DATMAN retrieval with existing data files.

A data file may have several physical records for each logical record (multiple cards per data row) and there may be physical records (cards) which are not desired to be accessed by a particular format file. The specification of which physical record within the logical record that a particular variable occurs on is given by PHYREC. The total number of physical records in one logical record is given by TOTREC. The maximum physical record size is 132 characters.

Within a particular physical record values are given in columns. LOWCOL and HGHCOL give the column boundaries within which to find the values. It is possible for a particular column or set of consecutive columns to contribute to several distinct variables. For example, DATE could be columns 10-15 and simultaneously DAY could be columns 10-11, MONTH columns 12-13, and YEAR columns 14-15.

The arguments to FMTBLD are:

FMTFIL - character - The name of the format file to be constructed.

LENG - INTEGER*2 - length of FMTFIL in characters.

VARNAM - INTEGER*2 array. Variable names to be described by the format file created. Names must be at most ten characters long with letters, numbers, $ and _ as valid characters. Each name is associated with a code when inserted into VARNAM and this code is used to associate the appropriate values of PHYREC, LOWCOL and HGHCOL with each variable. Created by DBVINS.

PHYREC - INTEGER*2 array which gives the physical record within a logical record of each variable. The Ith position in PHYREC is the record number of the variable given a code of I in VARNAM.

LOWCOL - Integer*2 array which gives the lower column within the physical record of the variables. The Ith position in LOWCOL is the low column for the variable given a code of I in VARNAM.

HGHCOL - INTEGER*2 array which gives the high column within the physical record of the variables. The Ith position in HGHCOL is the high column for the variable given a code if I in VARNAM.

TYPE - INTEGER*2 array. Used to pass the type of the variables.

The Ith position in TYPE is the type for the variable given a code of I in VARNAM. The values are:

1 - INTEGER

2 - Alphanumeric

3 - REAL

TOTREC - INTEGER*2. The total number of physical records in a logical record.

ERROR - INTEGER*2. Error indicator after call.

Values are:

0 - No error

1 - illegal variable name

2 - inconsistency between PHYREC and TOTREC or illegality in either

3 - inconsistency between LOWCOL and HGHCOL or illegality in either

4 - illegal type code

SFBLD (FILNAM, FILLEN, FMTFIL, FMTLEN, DATFIL, DATLEN)

SFBLD is a subroutine used to construct a searchable file usable by DBGET for retrieval. A searchable file consists of a data file (DATFIL) and a format file (FMTFIL) built by FMTBLD or the terminal user interface.

The data file is a fixed format ASCII file whose data values are described by the format file. It will usually be a disk file. If the data file is TERM then DBGET will ask the user at the terminal for values of the variables (by name) and thus the FORTRAN interface can also be used for interactive input.

The input to SFBLD is:

FILNAM - character - the name of the searchable file to be created.

FILLEN - INTEGER*2 length of FILNAM in characters.

FMTFIL - character - the name of the format file for the searchable file to be created.

FMTLEN - INTEGER*2 length of FMTFIL in characters.

DATFIL - character - the name of the fixed format ASCII file described by FMTFIL. If the value is TERM then the input is assumed to come from the terminal and the user is prompted for the input by variable name.

DATLEN - INTEGER*2 length of DATFIL in characters.

SECTION III - SAMPLE PROGRAMS

The first several examples will be retrieving data from the data base named ANALYT. It represents the sample situation of gathering samples at distinct locations (stations), assigning each sample an identification number and analyzing each sample for several metals.

The variables and their type for this data base are:

Variable	Type	Index Information
SNUM	Integer	Primary index variable
STATID	Alphanumeric	Index variable
LAT	Real	
LONG	Real	
MGCONC	Real	
PBCONC	Real	
ZNCONC	Real	

EXAMPLE OF DATA BASE ANALYT

SNUM	STATID	PBCONC	LAT	MGCONC	LONG
85602	ICEA	3.56	42.365		85.62
36805	ICEB		43.423		80.854
85603	ICEA	6.28	42.365		85.62
11000	2CEA	5.8	40.832		89.63
.	.	.	.		.
.	.	.	.		.
.	.	.	.		.

- SNUM is used to distinguish among rows (primary index)
- STATID is used to speed up retrieval (index)

Represents sample situation of taking samples at several locations (stations) and assigning each sample a number. Each sample is then analyzed for several metals.

Example program 1 does sequential retrieval of all rows in ANALYT.

```
      INTEGER*2 WORK(200),VALUE(100),VARNAM(30),LOWVAL(100),HGHVAL(100)
      REAL*8 V1,V2                    /* This will be for LAT and PBCONC
      INTEGER*4 IDEN                  /* This will hold SNUM
      INTEGER*2 LOC(5)                /* This will hold STATID
C
      CALL DBINIT(1)                   /* Initialize DATMAN
      CALL DBOPN('ANALYT',6,WORK,1,ERROR)
C
C     Open data base ANALYT for input
C     WORK is the controlling array for this
C     Data base for subsequent calls
C
      CALL DBVINT(VARNAM)              /* Initialize names array
      CALL DBVINS(VARNAM,1,'STATID',6)/* STATID will be given code of 1
      CALL DBVINS(VARNAM,2,'LAT',3)    /* LAT will have code of 2
      CALL DBVINS(VARNAM,3,'PBCONC',6)/* PBCONC will have code of 3
      CALL DBVINS(VARNAM,4,'SNUM',4)     /* SNUM will have code of 4
      CALL DBNAME(WORK,VARNAM,ERROR)     /* Make names known to data
                                            base
```

```
C
C    Now we can do retrieval
C
100 CONTINUE
    CALL DBGET(WORK,VALUE,EOFLG)
    IF(EOFLG .EQ. 1) Go to 200
C
C    We have gotten next record, now
C    we extract values from VALUE
C
    CALL DBVEX(VALUE,2,V1,LENG)       /* Get LAT into V1
    CALL DBVEX(VALUE,3,V2,LENG)       /* Get PBCONC into V2
    CALL DBVEX(VALUE,1,LOC,LENG)      /* Get STATID
    CALL DBVEX(VALUE,4,IDEN,LENG)     /* Get SNUM
    WRITE(1,900)LOC,IDEN,V1,V2
900 FORMAT(' ',5A2,I6,2F10.5)
    GO TO 100
200 CONTINUE
    CALL DBCLOS(WORK)
    STOP
    END
```

Example 1

Example program 2 represents retrieval from ANALYT of exactly those rows that have a station identifier of ICEA.

```
      INTEGER*2 WORK(200),VALUE(100),VARNAM(30),LOWVAL(100),HGHVAL(100)
      REAL*8 V1,V2                    /* This will be for LAT and PBCONC
      INTEGER*4 IDEN                  /* This will hold SNUM
      INTEGER*2 LOC(5)                /* This will hold STATID
C
      CALL DBINIT(1)                  /* Initialize DATMAN
      CALL DBOPN('ANALYT',6,WORK,1,ERROR)
C
C     Open data base ANALYT for input
C     WORK is the controlling array for this
C     Data base for subsequent calls
C
      CALL DBVINT(VARNAM)              /* Initialize names array
      CALL DBVINS(VARNAM,1,'STATID',6)/* STATID will be given code of 1
      CALL DBVINS(VARNAM,2,'LAT',3)    /* LAT will have code of 2
      CALL DBVINS(VARNAM,3,'PBCONC',6)/* PBCONC will have code of 3
      CALL DBVINS(VARNAM,4,'SNUM',4)   /* SNUM will have code of 4
      CALL DBNAME(WORK,VARNAM,ERROR)   /* Make names known to data
                                          base
      CALL DBVINT(LOWVAL)
      CALL DBVINS(LOWVAL,1,'ICEA',4)          /* Assign value to array
      CALL DBFIND(WORK,LOWVAL,LOWVAL,1,ERROR) /* Define subset, 1 means
                                                 new subset
```

```
C
C    Now we can do retrieval
C
100 CONTINUE
      CALL DBGET(WORK,VALUE,EOFLG)
      IF(EOFLG .EQ. 1) Go to 200
C
C    We have gotten next record, now
C    we extract values from VALUE
C
      CALL DBVEX(VALUE,2,V1,LENG)        /* Get LAT into V1
      CALL DBVEX(VALUE,3,V2,LENG)        /* Get PBCONC into V2
      CALL DBVEX(VALUE,1,LOC,LENG)       /* Get STATID
      CALL DBVEX(VALUE,4,IDEN,LENG)      /* Get SNUM
      WRITE(1,900)LOC,IDEN,V1,V2
900 FORMAT(' ',5A2,I6,2F10.5)
      GO TO 100
200 CONTINUE
      CALL DBCLOS(WORK)
      STOP
      END
```

Example program 2

Example program 3 represents retrieval from ANALYT of those rows that have a sample identification of between 80000 and 90000.

```
      INTEGER*2 WORK(200),VALUE(100),VARNAM(30),LOWVAL(100),HGHVAL(100)
      REAL*8 V1,V2                    /* This will be for LAT and PBCONC
      INTEGER*4 IDEN                  /* This will hold SNUM
      INTEGER*2 LOC(5)                /* This will hold STATID
C
      CALL DBINIT(1)                  /* Initialize DATMAN
      CALL DBOPN('ANALYT',6,WORK,1,ERROR)
C
C     Open data base ANALYT for input
C     WORK is the controlling array for this
C     Data base for subsequent calls
C
      CALL DBVINT(VARNAM)             /* Initialize names array
      CALL DBVINS(VARNAM,1,'STATID',6)/* STATID will be given code of 1
      CALL DBVINS(VARNAM,2,'LAT',3)   /* LAT will have code of 2
      CALL DBVINS(VARNAM,3,'PBCONC',6)/* PBCONC will have code of 3
      CALL DBVINS(VARNAM,4,'SNUM',4)  /* SNUM will have code of 4
      CALL DBNAME(WORK,VARNAM,ERROR)  /* Make names known to data
                                         base
      CALL DBVINT(LOWVAL)             /* INTL is integer *4 function
      CALL DBVINS(LOWVAL,4,INTL(80000),4)   /* Assign lower bound
      CALL DBVINT(HGHVAL)
      CALL DBVINS(HGHVAL,4,INTL(90000),4)   /* Assign upper bound
      CALL DBFIND(WORK,LOWVAL,HGHVAL,1,ERROR)  /* 1 Means new subset
```

```
C
C    Now we can do retrieval
C
100 CONTINUE
      CALL DBGET(WORK,VALUE,EOFLG)
      IF(EOFLG .EQ. 1) Go to 200
C
C    We have gotten next record, now
C    we extract values from VALUE
C
      CALL DBVEX(VALUE,2,V1,LENG)        /* Get LAT into V1
      CALL DBVEX(VALUE,3,V2,LENG)        /* Get PBCONC into V2
      CALL DBVEX(VALUE,1,LOC,LENG)       /* Get STATID
      CALL DBVEX(VALUE,4,IDEN,LENG)      /* Get SNUM
      WRITE(1,900)LOC,IDEN,V1,V2
900 FORMAT(' ',5A2,I6,2F10.5)
      GO TO 100
200 CONTINUE
      CALL DBCLOS(WORK)
      STOP
      END
```

Example program 3

Example program 4 represents the combination of the sampling situations of programs 2 and 3. That is, we wish to retrieve those rows with a sample identification between 80000 and 90000 and also a station identifier of ICEA.

```
      INTEGER*2 WORK(200),VALUE(100),VARNAM(30),LOWVAL(100),HGHVAL(100)
      REAL*8 V1,V2                    /* This will be for LAT and PBCONC
      INTEGER*4 IDEN                  /* This will hold SNUM
      INTEGER*2 LOC(5)                /* This will hold STATID
C
      CALL DBINIT(1)                  /* Initialize DATMAN
      CALL DBOPN('ANALYT',6,WORK,1,ERROR)
C
C     Open data base ANALYT for input
C     WORK is the controlling array for this
C     Data base for subsequent calls
C
      CALL DBVINT(VARNAM)             /* Initialize names array
      CALL DBVINS(VARNAM,1,'STATID',6)/* STATID will be given code of 1
      CALL DBVINS(VARNAM,2,'LAT',3)   /* LAT will have code of 2
      CALL DBVINS(VARNAM,3,'PBCONC',6)/* PBCONC will have code of 3
      CALL DBVINS(VARNAM,4,'SNUM',4)  /* SNUM will have code of 4
      CALL DBNAME(WORK,VARNAM,ERROR)  /* Make names known to data base
      CALL DBVINT(LOWVAL)             /* INTL is integer *4 function
      CALL DBVINS(LOWVAL,4,INTL(80000),4)   /* Assign SNUM lower bound
```

```
      CALL DBVINS(LOWVAL,1,'ICEA',4)          /* Assign STATID value
      CALL DBVINT(HGHVAL)
      CALL DBVINS(HGHVAL,1,'ICEA',4)          /* Assign STATID
      CALL DBVINS(HGHVAL,4,INTL(90000),4)     /* Assign SNUM upper value
      CALL DBFIND(WORK,LOWVAL,HGHVAL,1,ERROR) /* Define new subset

C
C     Now we can do retrieval
C
100 CONTINUE
      CALL DBGET(WORK,VALUE,EOFLG)
      IF(EOFLG .EQ. 1) Go to 200
C
C     We have gotten next record, now
C     we extract values from VALUE
C
      CALL DBVEX(VALUE,2,V1,LENG)        /* Get LAT into V1
      CALL DBVEX(VALUE,3,V2,LENG)        /* Get PBCONC into V2
      CALL DBVEX(VALUE,1,LOC,LENG)       /* Get STATID
      CALL DBVEX(VALUE,4,IDEN,LENG)      /* Get SNUM
      WRITE(1,900)LOC,IDEN,V1,V2
900 FORMAT(' ',5A2,I6,2F10.5)
      GO TO 100
200 CONTINUE
      CALL DBCLOS(WORK)
      STOP
      END
```

Example program 4

Example program 5 represents a different situation than the prior examples and is the most complicated example. This example is motivated by the mapping of digitized data.

Suppose the boundary of each state has been digitized. Each state has a distinct data base that contains this data. Each data base has the following variables. Each row will represent one point of digitized data.

Variable	Purpose
LAT	The latitude of the point.
LON	The longitude of the point.
LINNUM	Each continuous line has a distinct identifier. This allows shore lines to be distinguished from islands and from interior boundaries.
COUNT	Each point of each line is assigned a number. The line is drawn by accessing these points in increasing order.

The last two of these variables are index variables.

Now suppose that a portion, the East Coast, is to be mapped. Then we would establish a separate data base that would control the accessing of the digitized data. This data base has a name DEMO. The variables of DEMO ARE:

Variable	Purpose
DBNAME	The name of the digitized data base described by this row.
LINNUM	The particular line desired described by this row.
SCALE	The retrieval rate of the digitized data.

A program to retrieve the latitude and longitude of the points to be plotted to give the desired portion of the coast is example 5.

```
      INTEGER*2 WORK1(200),WORK2(200),VALUE1(100),VALUE2(100)
      INTEGER*2 NAMES(100)
      REAL*8 ALAT,ALON
      INTEGER*2 DBNAM(5)
      INTEGER*4 ITEMP
      REAL*8 DTEMP
C
      CALL DBINIT(1)
C
C     First we open the data base DEMO
C
      CALL DBOPN('DEMO',4,WORK1,1,ERROR)
C
C     Now we define the names we want from
C     DEMO
C
      CALL DBVINT(NAMES)
      CALL DBVINS(NAMES,1,'DBNAME',6)  /* Code 1 from DEMO is the
                                          name of data base of
                                          digitized data
      CALL DBVINS(NAMES,2,'LINNUM',6)  /* Code 2 from DEMO is the
                                          line number desired
      CALl DBVINS(NAMES,3,'SCALE',5)   /* Code 3 from DEMO is the
                                          scale desired
      CALL DBNAME(WORK1,NAMES,ERROR)
```

```
C
C   Now we reuse NAMES to prepare to
C   define codes for digitized data base
C
      CALL DBVINT(NAMES)
      CALL DBVINX(NAMES,1,'LAT',3)      /* LAT will have code 1 in
                                           data base of digitized
                                           data
      CALL DBVINS(NAMES,2,'LON',3)      /* LON will have code 2
      CALL DBVINS(NAMES,3,'COUNT',5)
      CALL DBVINS(NAMES,4,'LINNUM',6)
C
C   Now we are ready to begin major loop
C
100 CONTINUE
C
C   Retrieve next row from DEMO
C
      CALL DBGET(WORK1,VALUE1,EOFLG)
      IF(EOFLG .EQ. 1) Go to 200
C
C   Get the data base name
C
      CALL DBVEX(VALUE1,1,DBNAM,LENG)
C
C   Now open digitized data base
C
      CALL DBOPN(DBNAM,LENG,WORK2,1,ERROR)
```

```
C
      CALL DBNAME(WORK2,NAMES,ERROR)
C
C     Now set scale
C
      CALL DBVEX(VALUE1,2,DTEMP,LENG)
      TEMP=DTEMP                              /* Convert to single
                                                 precision
      CALL DBSCAL(WORK2,TEMP,ERROR)           /* Set scale
C
C     Now retrieve the line number
C
      CALL DBVEX(VALUE1,2,ITEMP,LENG)
C
C     Now we define the subset consisting
C     of that line number
C
      CALL DBVINT(VALUE2)
      CALL DBVINS(VALUE2,4,ITEMP,LENG)
      CALL DBFIND(WORK2,VALUE2,VALUE2,1,ERROR)
C
C     Finally we retrieve in increasing order
C     of sequencing count
C
      CALL DBORDR(WORK2,3,ERROR)
C
C     Now we retrieve LAT and LON from data base
140   CONTINUE
```

```
      CALL DBGET(WORK2,VALUE1,EOFLG)
      IF(EOFLG .EQ. 1) Go to 160
C
C     Pick out LAT and LONG and for
C     lack of better thing to do, write
C     them out
C
      CALL DBVEX(VALUE1,1,ALAT,LENG)
      CALL DBVEX(VALUE1,2,ALON2,LENG)
      WRITE(1,999)ALAT,ALON
999   FORMAT(2F15.5)
      GO TO 140
160   CONTINUE
C
C     We get here when exhausted digitized data
C     from one data base, get next one
C
      CALL DBCLOS(WORK2)
      GO TO 140
200   CONTINUE
C
C     Exhausted all lines
C
      CALL DBCLOS(WORK1)
      STOP
      END
```

Example Program 5

SECTION IV - APPENDIX

APPENDIX A

DATA STRUCTURE OF DATA BASE AND NAMES DATA BASE

The structure of one logical record in a data base will be discussed. Since each logical record in a data base is independent of any other logical record, they are all of the same structure.

A logical record in a data base may consist of one or more physical records. The maximum physical record size used in DATMAN is 1000 words. The actual physical record size used is determined by the RECSIZ parameter to DBBLD. If RECSIZ is less than or equal to 1000 then this is the actual physical record size. If RECSIZ is larger than 1000 then the number of physical records required per logical record is determined by dividing RECSIZ by 1000 and rounding up. Once the number of physical records per logical record has been determined the physical record size is calculated by requiring all physical records to be the same size. Thus, the physical record size can be determined by dividing RECSIZ by the number of physical records per logical record and rounding up.

A variable value within a logical record is determined by:

1) physical record number within logical record
2) low column of variable
3) high column of variable

DATMAN places and extracts values within column boundaries of the physical records. Integer values take up 4 columns, real values 8 bytes and alpha variables the length passed in VARLEN during creation of the data base.

The variables of a data base are controlled by a data base called nameDD where name is the name of the original data base. The variables of nameDD are:

1) NAMES - variable name - alpha length 10
2) COL1 - low column of variable - INTEGER
3) COL2 - high column of variable - INTEGER
4) INDCOD - index code - INTEGER
   - 0 - not index value
   - 1 - primary index
   - 2 - secondary index
5) TYPE - type of variable - INTEGER
   - 1 - integer (4 bytes)
   - 2 - alpha (COL2-COL1+1 bytes)
   - 3 - real (8 bytes)
6) COMMENT - description of variable - alpha length 80
7) LRECL - physical record number within one logical record - INTEGER

To add a new variable to data base name use DBADD on data base nameDD and furnish values. To delete a variable use DBDELT on nameDD with LOWVAL and HGHVAL specified in terms of NAMES. To change a characteristic of a variable use DBCHNG with LOWVAL and HGHVAL specified in terms of NAMES.

CAUTION

It is possible to destroy integrity of a data base by improper modification of the names data base. Some specific items of which to beware are:

1. Changing type of variable. Numeric values are stored in binary and hence any data in a numeric field would be invalid if interpreted as characters or vice versa. Numeric values are also interpreted as either integer or real depending upon their assigned type and any values would be incorrectly evaluated if the wrong type were chosen.
2. Changing column values of existing variables. This would result in incorrect values for those variables.
3. Changing any items of an index variable or making a new variable an index variable. If any changes are made to any index value (e.g., change of index code) then the terminal interface INDEX must be run to reconstruct the index files.

APPENDIX B

ACCESSING SUBROUTINES ON PRIME

The FORTRAN subroutines exist in two forms on the PRIME. These are as R-Mode subroutines and as V-Mode subroutines. To use the R-Mode subroutines in the input to LOAD, put LI DATLIB followed by LI APPLIB. Ignore warnings that result. To use the V-Mode subroutines in the input to SEG, put LI VDATLIB followed by LI VAPPLB.

APPENDIX C

ACCESSING SUBROUTINES ON PDP 11/70

DATMAN is scheduled to be available on PDP 11/70 under IAS on 1/1/81.

APPENDIX D

SUBROUTINES AND FUNCTIONS

DBADD	add data to existing data base
DBBLD	create new data base
DBCLOS	close open data base
DBCHNG	change data in existing data base
DBDELT	delete rows from existing data
DBFIND	define subset for retrieval
DBGET	retrieve next row of data
DBINDX	returns index code of specified variable
DBINIT	initialize DATMAN system
DBJOIN	join two (or more) physical data bases for retrieval
DBLENG	returns length of specified variable
DBNAME	make known set of names for manipulation

DBOPN	open data base for manipulation
DBORDR	define order of retrieval
DBREWD	rewind open data base
DBSCAL	set scale for retrieval
DBTYPE	return type of specified variable
DBVEX	extract value from DATMAN value array
DBVINS	insert value into DATMAN value array
DBVINT	initialize DATMAN value array
FMTBLD	create DATMAN format file
SFBLD	create searchable file for use in subsequent retrievals